**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**

**“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. ІГОРЯ СІКОРСЬКОГО”**

**НАВЧАЛЬНО НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ**

Комп’ютерна графіка

Лабораторна робота №5

Виконав:

Студент 2 курсу

Групи ФІ-21

Поштаренко Сергій

Київ 2024

# ««Методи інтерполювання зображень, цифровий зум»

***Мета роботи***: опанувати сучасні методи інтерполювання зображень, які

є основою цифрового зуму різних графічних елементів, навчитись визначити

переваги і недоліки кожного з методів інтерполювання у ході застосування даних

методів конкретно для зображень.

***Завдання:***

1. Ознайомитись з теоретичними відомостями лабораторної роботи щодо

виконання інтерполювання зображень з метою здійснення збільшення

зображень і графічних елементів загалом.

1. Вивчити з математичної точки зору усі відомі методи інтерполювання, що

наведені у даній лабораторній роботі.

1. Після вивчення математичних методів програмно реалізувати

інтерполювання функції (обрати самостійно) виду z = f(x, y) різними

методами, а саме: лінійне інтерполювання, поліноміальне інтерполювання,

інтерполювання з використанням розкладу в ряд Фур’є, інтерполювання з

використанням методу найменших квадратів. Продемонструвати отримані

результати.

1. За основу розробки власних програмних реалізацій за бажанням можна

брати наведені програмні коди.

1. Програмно реалізувати інтерполювання зображення (взяти маленьке

зображення і порівняно велике зображення) різними методами, а саме:

лінійне інтерполювання, поліноміальне інтерполювання, інтерполювання

з використанням розкладу в ряд Фур’є, інтерполювання з використанням

методу найменших квадратів. Продемонструвати отримані результати.

Визначити переваги і недоліки кожного з реалізованих методів.

***Результат:***

import numpy as np

import matplotlib.pyplot as plt

from PIL import Image

from IPython.display import display

class Interpolation:

    def \_\_init\_\_(self, type:str ="image", \*args):

        if type == "image":

            self.image = Image.open(args[0])

        elif type == "function":

            self.function = args[0]

    def linear\_interpolation(self, \*\*kwargs):

        my\_image = np.asarray(self.image)

        m, n, k = my\_image.shape

        my\_image\_new = np.zeros((2\*m-1, 2\*n-1, k))

        my\_image\_new[:2\*m-1:2, :2\*n-1:2, :] = my\_image

        my\_image\_new[1:2\*m-1:2, :2\*n-1:2, :] = 0.5 \* (my\_image\_new[:2\*m-3:2, :2\*n-1:2, :] + my\_image\_new[2:2\*m-1:2, :2\*n-1:2, :]) # Down

        my\_image\_new[:2\*m-1:2, 1:2\*n-1:2, :] = 0.5 \* (my\_image\_new[:2\*m-1:2, :2\*n-3:2, :] + my\_image\_new[:2\*m-1:2, 2:2\*n-1:2, :]) # Right

        my\_image\_new[1:2\*m-1:2, 1:2\*n-1:2, :] = 0.5 \* (my\_image\_new[:2\*m-3:2, :2\*n-2:2, :] + my\_image\_new[2:2\*m-1:2, :2\*n-2:2, :]) # Middle

        self.image\_modified = np.uint8(my\_image\_new)

        if "save" in kwargs and kwargs["save"] == True:

            if "path" in kwargs:

                Image.fromarray(self.image\_modified).save(kwargs["path"])

            else:

                Image.fromarray(self.image\_modified).save("1.jpg")

        if "show" in kwargs and kwargs["show"] == True:

            display(Image.fromarray(self.image\_modified))

        if "replace" in kwargs and kwargs["replace"] == True:

            self.image = self.image\_modified

        if "rvalue" in kwargs and kwargs["rvalue"] == True:

            return self.image\_modified

        return self

    def polynomial\_interpolation(self):

        nx = self.image.size[0]

        ny = self.image.size[1]

        my\_image = np.array(self.image)

        my\_image\_new = np.zeros\_like(my\_image)

        kx = 1

        for Lx in range(1, nx):

            xx = []

            for i in range(1, N\_step\_polinom+1):

                xx = [xx, X[kx]]

                kx = kx + 1

            kx = kx - 1

            ky = 1

            for Ly in range(1, ny):

                yy = []

                for j in range(1, N\_step\_polinom+1):

                    yy = [yy, Y[ky]]

                    ky = ky + 1

                ky = ky - 1

                sol = build\_system(xx,yy)

                for i1 in range(xx[1], xx(N\_step\_polinom+1)):

                    for j1 in range(yy[1], yy(N\_step\_polinom+1)):

                        my\_image\_new[i1, j1,:] = polinom\_summ(sol, (i1-xx[1]+1)/xx[end], (j1-yy[1]+1)/yy[end])

        self.image\_modified = Image.fromarray(my\_image\_new)

        self.image\_modified.show()

    def furies\_interpolation(self):

        my\_image = np.array(self.image)

        my\_image\_new = np.zeros\_like(my\_image)

        nx = 30

        ny = 30

        x = np.arange(1, nx)

        y = np.arange(1, ny)

        put\_x\_pixel = 3

        put\_y\_pixel = 3

        for i in range(1, nx):

            X[i] = i+(i-1)\*put\_x\_pixel

            for j in range(1, ny):

                Y[j] = j+(j-1)\*put\_y\_pixel

                my\_image[i,j] = np.sin(x[i]/nx\*np.pi)\*np.cos(2\*y[j]/ny\*np.pi)+1

        [Y,X] = np.meshgrid(Y,X)

        nx\_new = nx+(nx-1)\*put\_x\_pixel

        ny\_new = ny+(ny-1)\*put\_y\_pixel

        x\_new = np.arange(1, nx\_new)

        y\_new = np.arange(1, ny\_new)

        [y\_new,x\_new] = np.meshgrid(y\_new,x\_new)

        my\_image\_new = resample(my\_image, put\_x\_pixel+1, 1)

        my\_image\_new = resample(my\_image\_new, put\_y\_pixel+1, 1)

        surf(x\_new, y\_new, my\_image\_new[1:nx\_new,1:ny\_new])

        fig = plt.figure()

        ax = fig.add\_subplot(111, projection='3d')

        ax.plot(X,Y,my\_image,'\*m')

        ax.set\_xlabel('X')

        ax.set\_ylabel('Y')

        ax.set\_zlabel('Z')

        plt.show()

    def least\_squares\_interpolation(self):

        # my\_image = np.array(self.image)

        # my\_image\_new = np.zeros\_like(my\_image)

        # N = 20

        # x = np.linspace(N, 1)

        # a = 2.1

        # b = -7.5

        # c = 10

        # y = a\*x\*\*2 + b\*x + c + np.linspace(N, 1)

        # A = np.array([x \*\* 2, x, np.ones((N, 1))])

        # b = y

        # b = A.T\*b

        # A = A.T\*A

        # sol = A/b

        # X = sort(x)

        # Y\_model = sol[1] \* X \*\* 2 + sol[2] \* X + sol[3]

        # plt.plot(x, y, ".")

        # plt.plot(X, Y\_model, ".")

        nx = 10 # кількість базових вузлів сітки по осі ОХ

        ny = 10 # кількість базових вузлів сітки по осі ОУ

        N\_step\_polinom = 4 # максимальна степінь (вище не застосовують)

        Nx = nx \* N\_step\_polinom+1

        Ny = ny \* N\_step\_polinom+1

        x = np.arange(1, nx)

        y = np.arange(1, ny)

        put\_x\_pixel = 3 # кількість пікселів, які необхідно вставити вздовж осі абсцис (більше не використовують)

        put\_y\_pixel = 3 # кількість пікселів, які необхідно вставити вздовж осі ординат (більше не використовують)

        for i in range(1, Nx):

            for j in range(1, Ny):

                my\_image[i,j] = np.sin(x[i]/Nx\*np.pi)\*np.cos(2\*y[j]/Ny\*np.pi)+2

        nx\_new = Nx+(Nx-1)\*put\_x\_pixel

        ny\_new = Ny+(Ny-1)\*put\_y\_pixel

        x\_new = np.arange(1, nx\_new)

        y\_new = np.arange(1, ny\_new)

        my\_image\_new = np.zeros(nx\_new,ny\_new)

        for i in range(1, Nx):

            X[i] = i+(i-1)\*put\_x\_pixel

            for j in range(1, Ny):

                Y[j] = j+(j-1)\*put\_y\_pixel

                my\_image\_new[X(i), Y(j)] = my\_image(i,j)

        kx = 1

        for Lx in range(1, nx):

            xx = []

            for i in range(1, N\_step\_polinom+1):

                xx = [xx, X[kx]]

                kx = kx + 1

            kx = kx - 1

            ky = 1

            for Ly in range(1, ny):

                yy = []

                for j in range(1, N\_step\_polinom+1):

                    yy = [yy, Y[ky]]

                    ky = ky + 1

                ky = ky - 1

                sol = build\_system(xx,yy)

                for i1 in range(xx[1], xx[N\_step\_polinom+1]):

                    for j1 in range(yy[1], yy[N\_step\_polinom+1]):

                        my\_image\_new[i1, j1] = polinom\_summ(sol,(i1-xx[1]+1)/xx[end],(j1-yy(1)+1)/yy[end])

        mesh(x\_new, y\_new, my\_image\_new)

        for i in range(1, Nx):

            for j in range(1, Ny):

                fig = plt.figure()

                ax = fig.add\_subplot(111, projection='3d')

                ax.plot(X[i], Y[j],my\_image[i,j],'m\*')

                ax.set\_xlabel('X')

                ax.set\_ylabel('Y')

                ax.set\_zlabel('Z')

                plt.show()

        size(my\_image\_new)

    def show(self):

        plt.imshow(Image.fromarray(self.image\_modified))

        plt.show()

image = Interpolation("image", "1.png.png")

image.linear\_interpolation().show()

image.linear\_interpolation(replace=True).linear\_interpolation(replace=True).linear\_interpolation(show=True)

Зображення, що містить мультфільм, знімок екрана

Автоматично згенерований опис

Зображення, що містить мультфільм, ілюстрація, Мультфільм, картинки

Автоматично згенерований опис